OTT Subscription Management System (Salesforce CRM)

# Phase 1: Project Initiation & Detailed Requirement Gathering

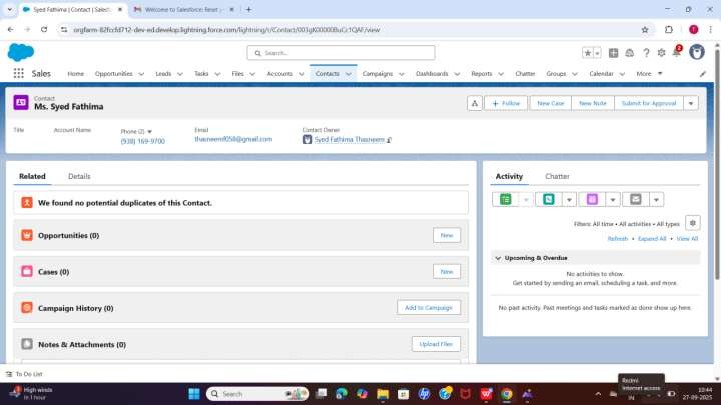
 Goal: Define the project scope, assemble the team, and collect granular requirements across all subscriber lifecycle stages (Acquisition, Service, Billing).

1. Conduct a project Kick-off Meeting to produce the Signed Project Charter (confirming scope, budget, timeline, and key stakeholders).
2. Run Deep-dive Workshops with stakeholders (Billing, Support, and Marketing teams) to create a Detailed Requirements Document.
3. Perform a System Audit to produce a Current State Analysis documenting existing systems (e.g., billing gateway, website) that require integration.

# Phase 2: Data Model and Technical Design (Configuration Setup)

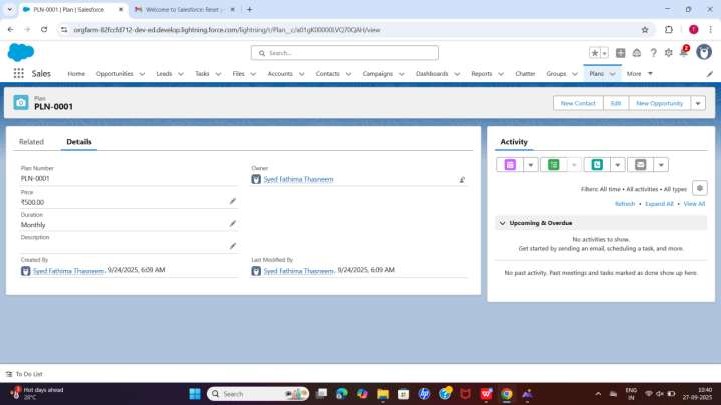
 Goal: Finalize the core data structure and design the overall system architecture by setting up the foundational objects and relationships.

1. Objects Created and Configured:
   * Contact (Standard Object): Stores customer details.



 Plan\_c (Custom Object): Represents subscription plans available.

* + Fields: Plan Number (Auto Number), Price (Currency INR), Duration (Picklist: Monthly, Quarterly, Yearly), Description (Text Area).

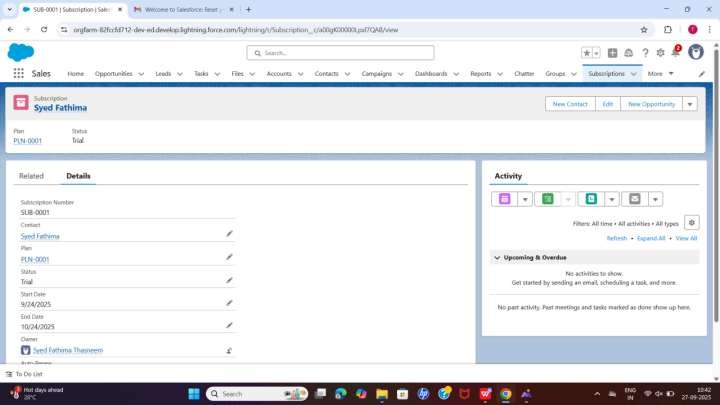


 Subscription\_c (Custom Object): Connects a Contact with a Plan.

* + Fields: Subscription Number (Auto Number), Contact (Lookup Contact),

Plan (Lookup — Plan\_c), Status (Picklist: Active, Inactive, Cancelled), Start

Date (Date), End Date (Date), Owner (Standard Owner field).



1. Relationships Established:



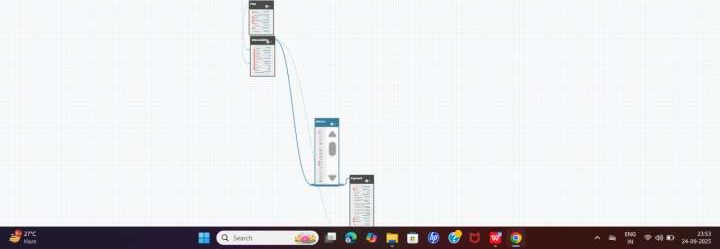
Contact

Subscription:

Lookup

relationship

established.



1. Demo Records Created (Verification):

 A Contact Record (Name: Syed Fathima) was created.

0 A Plan Record (Name: Premium Plan, Number: PLN-OOOI, Price: 500 INR, Duration: Monthly) was created.

@ OTTsubscription prr x PLN-OOOI I Plan I sa; x M Welcome to Salesfor X @ New Tab X Google Gemini u only provide me ir X Scheduled Subscrjpl

0-; orgfarm-82fccfd712-dev-eddevelop.lightning.force.com/lightning/r/Plan c/a01gK00000LVQ70QAH/view



Sales Home Opportunities v Leads v Tasks v Files v Accounts v Contacts v Campaigns V Dashboards V Reports V Chatter Groups v Plans V More

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Plan  PLN-OOOI | |  | |  |  |  | | --- | --- | --- | | New Contact | Edit | New Opportunity | |
| Related  Plan Number  PLN-OOOI  Price  \*500.00  Duration  Monthly  Description  Created By  Syed Fathima | Details  Thasneem, 9/24/2025, 6:09 AM | Owner  Syed Fathima Thasneem  Last Modified By  O syed Fathima Thesneem, 9/24/2025, 6:09 AM | Try Out the Compact Setting |
| See more without scrolling |
| Salesforce user research found that users overwhelmingly prefer the Compact density setting because it shows 30% more content on the page. You'll have all the information you need at a glance, without the extra whitespace.  Try It Out |
|  | |  |

To Do List

6' Trending videos ENG 19:49

Q Search

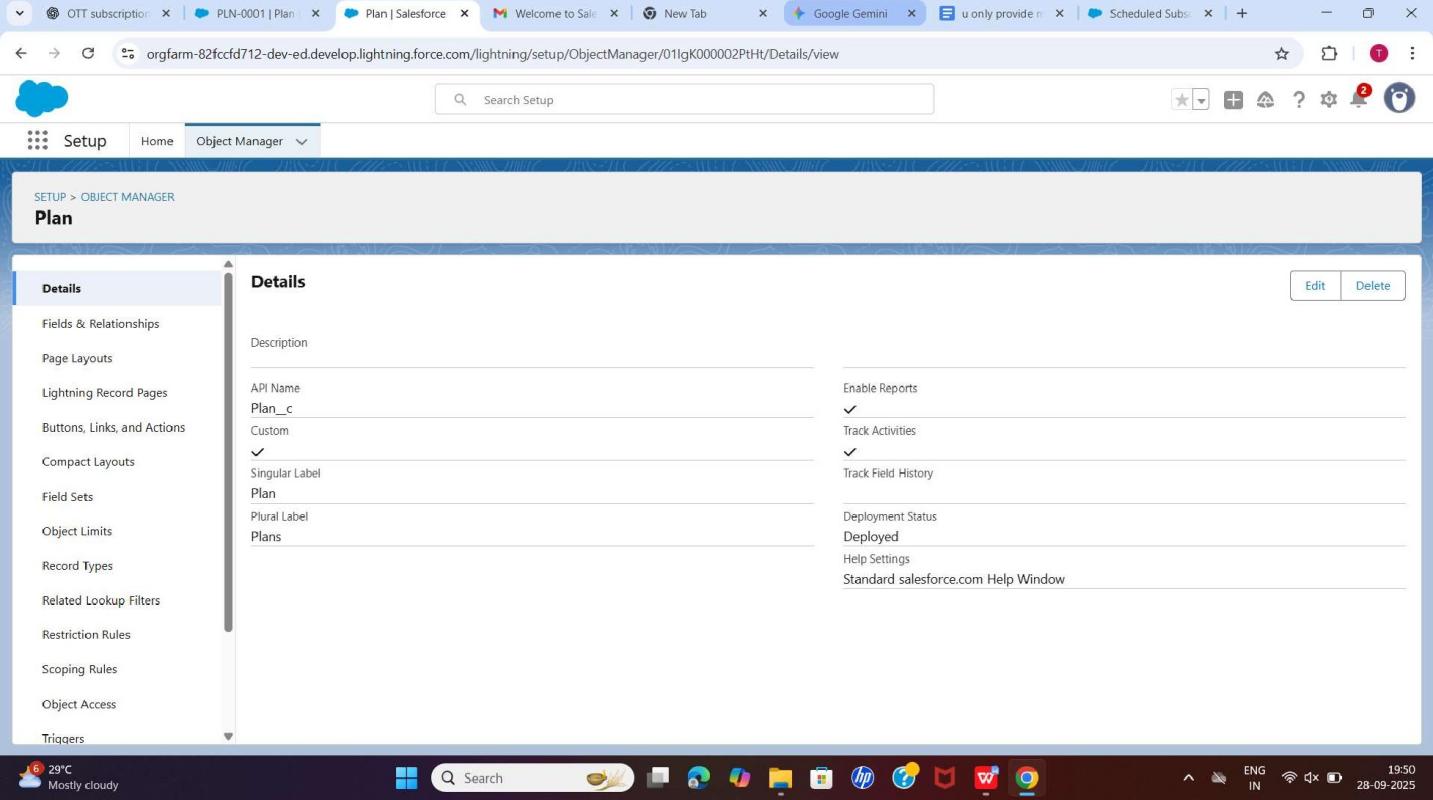
The Mandaloria... IN 28-09-2025  A sample Subscription Record (GEN-2004-001234) was created and linked to the Contact and Plan, with Status Active.

1. Functional Steps Completed:
   * Created a Contact and a Plan (Product) in Salesforce.
   * Created a Subscription record linked to the Contact and Plan.
   * Verified the lookup relationships between Subscription Contact and Subscription  plan.
   * Confirmed the subscription is functional and saved correctly.
2. Notes: The initial setup for the core objects is complete, with Price entered in Indian Rupees (INR) and the configuration verified.

# Phase 3: Data Modeling & Relationships (Billing Structure)

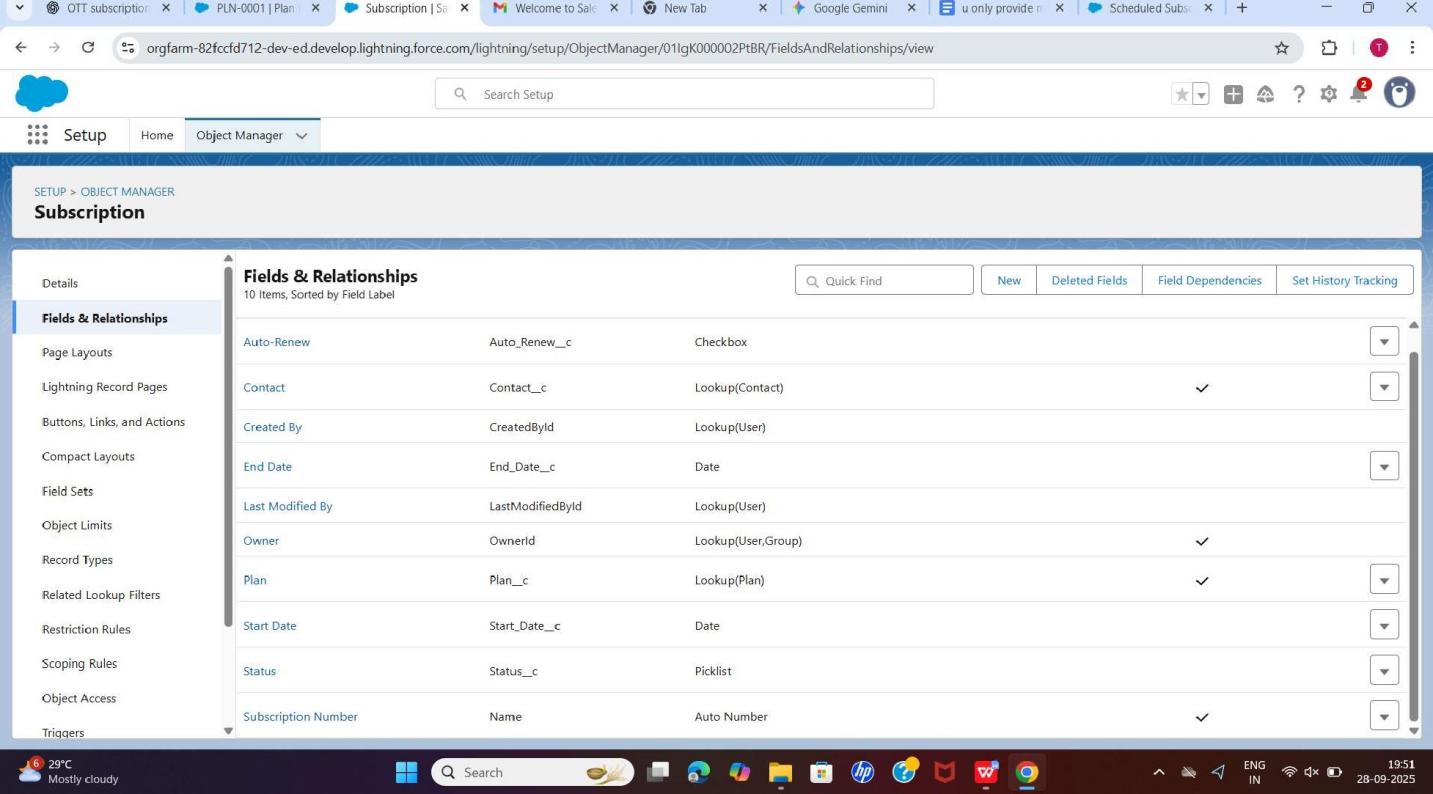
 Goal: Build the complete data structure in Salesforce, including transactional objects, and establish all necessary Master-Detail relationships for billing.

1. Objects Created (Confirmed & Added): o Standard Object: Contact (Subscriber).
   * Custom Objects (Transactional): Subscription c, Plan c, Payment c, Refund c.



1. Fields Defined (Detailed):

 Subscription c: Subscription Number (Auto Number), Contact c (Lookup to Contact), Plan c (Lookup to Plan), Start Date (Date), End Date (Date), Status (Picklist: Active, Cancelled, Paused, Expired), Auto-Renew c (Checkbox).



* + Plan c: Plan Number (Auto Number), Name (Text), Price c (Currency, INR), Duration\_c (Picklist: Monthly, Quarterly, Yearly), Description c (Rich Text).

 Payment c: Payment Number (Auto Number), Amount (Currency, INR), Date

(Date/Time), Status (Picklist: Success, Failed, Pending, Refunded), Subscription c (Master-Detail), Total Refund Applied / Total Refund Unapplied (Roll-Up Summary).

 Refund c: Refund Number (Auto Number), Amount (Currency, INR), Date (Date/Time), Status (Picklist: Processed, Pending), Payment c (Master-Detail), Subscription c (Lookup, optional).

1. Record Types:

O Plan c Record Types: Created Monthly Plan and Yearly Plan.

1. Page Layouts and Compact Layouts:
   * Subscription c Layout: Includes related lists for Payments and Refunds. Fields displayed: Contact, Plan, Status, Start Date, End Date, Auto-Renew.
   * Subscription Compact Layout: Set as Primary, displaying Contact (Subscriber), Plan, and Status.
2. Relationships (Schema Confirmed):
   * Subscription c to Contact\_c (Type: Lookup). o Subscription c to Plan c (Type: Lookup).
   * Payment c to Subscription c (Type: Master-Detail). o Refund c to Payment c (Type: Master-Detail).
3. Schema Builder: All objects (Contact, Subscription c, Plan c, Payment c, Refund c) and their relationships are visually confirmed.
4. Status: Phase 3 Complete: All objects, fields, record types, page layouts, compact layouts, and relationships have been created and verified.

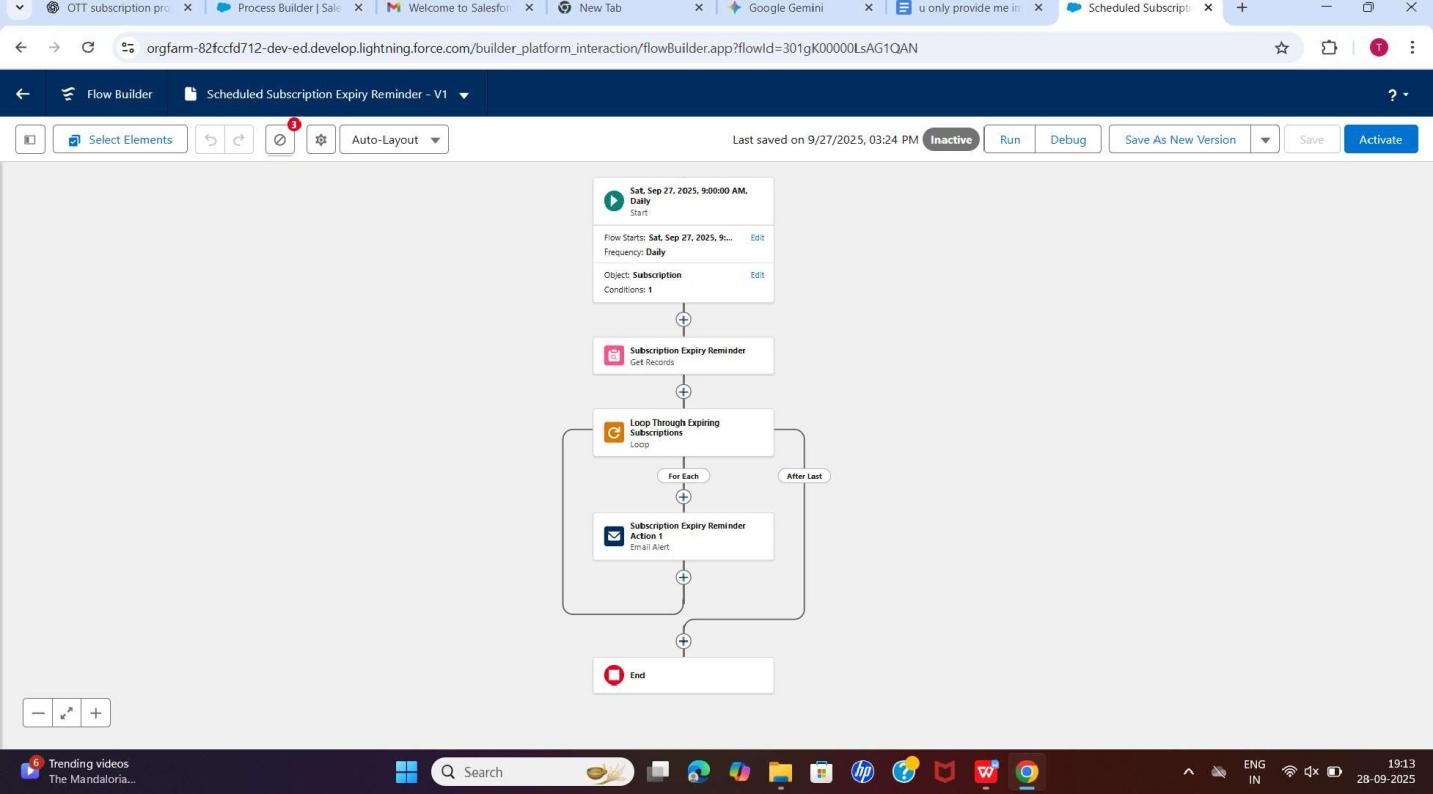
# Phase 4: Sandbox Development (Configuration & Customization)

 Goal: Build all custom objects, fields, and automation in a dedicated development sandbox environment, and provide supporting evidence (screenshots).

1. Custom Object Creation Confirmation:
   * All custom objects and fields from Phase 2 and 3 are deployed to the development sandbox.
   * Evidence: Screenshot of the Object Manager confirming the status of all custom objects.

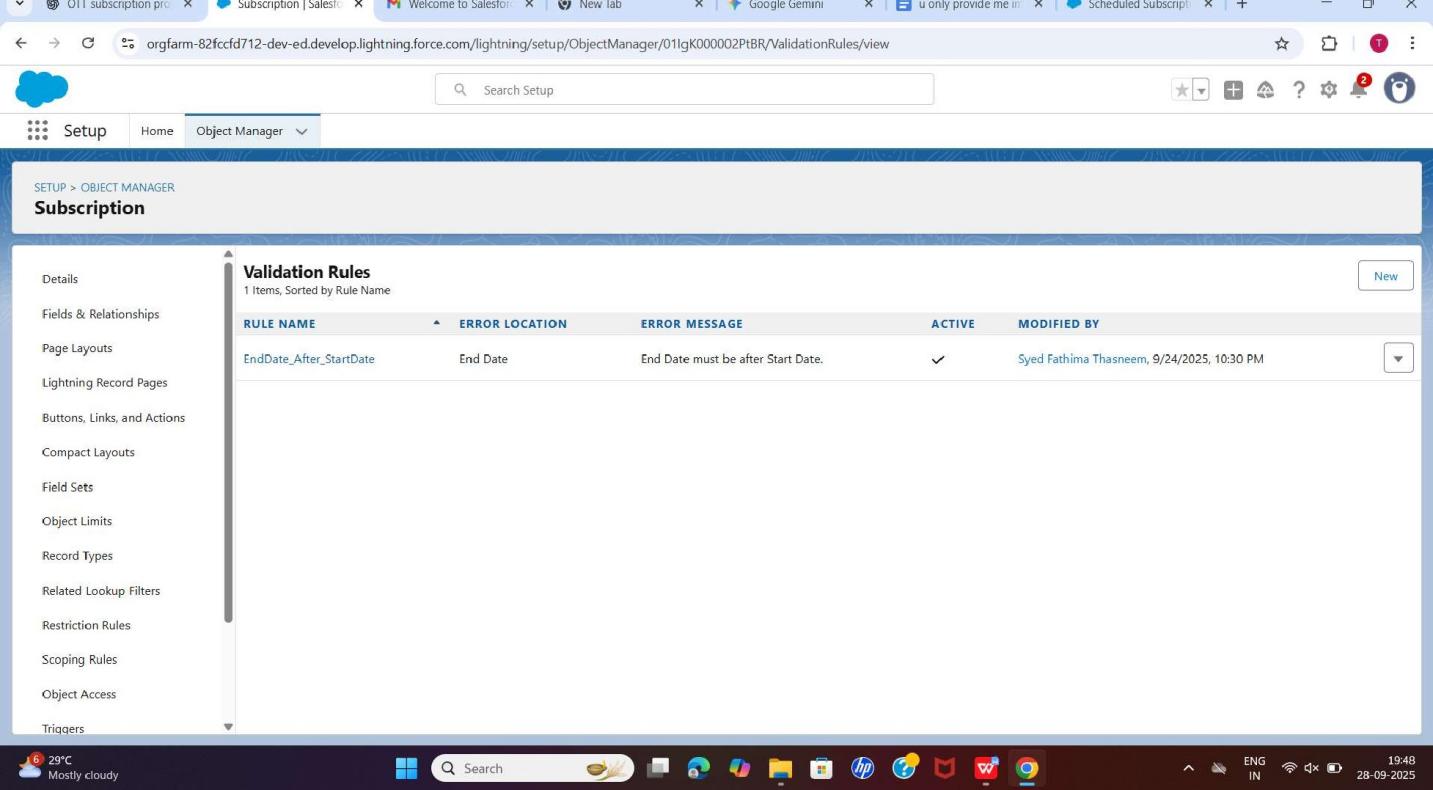
[Image 01] (Screenshot of Custom Objects in Setup)

1. Automation Builds (Flows & Rules):
   * Flow: Auto\_Subscription\_Creation (Record-Triggered Flow): Automatically creates a new Subscription c record when a new Contact is created and meeting specific criteria (e.g., Lead Source = 'Website Signup').



 Flow: Payment\_Status\_Updater (Record-Triggered Flow): Updates the Subscription c.Status field based on the related Payment c status (e.g., Payment Status 'Success' updates Subscription Status to 'Active').

 Validation Rule: Valid \_ Refund \_Amount (Payment c): Prevents a user from creating a Refund c amount that exceeds the original Payment c.Amount using a formula rule.



@

X

M

X

@

X

X

X

* + Evidence: Screenshots of the Flow Builder and the Validation Rule definition screen.

[Image 03] (Screenshot of the Valid\_Refund\_Amount Validation Rule on the Payment c object)

1. Service Console Setup & Quick Action:
   * App Setup: Configured the Service Console app with the appropriate navigation items (Contacts, Subscriptions, Payments, Cases).
   * Screen Flow: Built and deployed the Quick Action: Reset\_Password Screen Flow for support agents to use directly from the Contact record page.
   * Evidence: Screenshots of the configured Service Console app and the quick action in use on a record page.
2. User Access Setup (Profiles):
   * Created and configured the Profile: OTT\_Support\_Agent and Profile: Content\_Manager.
   * Permissions configured to allow Support Agents Read/Edit/Create access to

Subscription c and Payment c, while Content Managers have Read-Only access to financial objects.

* + Evidence: Screenshot of the Profile definition page showing object access for OTT\_Support\_Agent.

[Image 06] (Screenshot of Profile Settings for OTT\_Support\_Agent)

# **OTT Subscription System – Phase 5:**

# **📌 Phase 5 Overview**

# In Phase 5, we implemented **advanced business logic** in the OTT Subscription System using **Apex classes and triggers**.

# **Goals of this phase:** 1. Prevent overlapping bookings for OTT subscriptions. 2. Use **Trigger Handler classes** for reusable and clean logic. 3. Ensure **bulk-safe operations** for multiple inserts/updates. 4. Write **unit tests** to cover all scenarios for deployment.

# **🎯 Objects & Fields**

# \*\*Booking\_\_c Object\*\*

|  |  |  |  |
| --- | --- | --- | --- |
| Field Label | Field Name | Data Type | Notes |
| Booking Number | Name | Auto Number | System-generated |
| Start Date | Start\_Date\_\_c | Date | Required for booking logic |
| End Date | End\_Date\_\_c | Date | Required for booking logic |
| Status | Status\_\_c | Picklist | Active / Inactive |
| Total Amount | Total\_Amount\_\_c | Currency | Required for billing |
| Discount | Discount\_\_c | Currency | Optional |
| Owner | OwnerId | Lookup(User/Group) | System managed |
| Created By | CreatedById | Lookup(User) | System managed |
| Last Modified By | LastModifiedById | Lookup(User) | System managed |

# 

# **⚡ Apex Classes**

# **1. BookingTriggerHandler**

# Handles the **core logic** of the Booking trigger.

# Ensures **no overlapping bookings**.

# Bulk-safe for multiple records.

# 

# 

# 

# 

# 

# 

# 

# public class BookingTriggerHandler { public static void handleBeforeInsertUpdate(List<Booking\_\_c> newBookings) { List<Booking\_\_c> existing = [SELECT Id, Start\_Date\_\_c, End\_Date\_\_c FROM Booking\_\_c]; for (Booking\_\_c nb : newBookings) { for (Booking\_\_c eb : existing) { if (nb.Id != null && nb.Id == eb.Id) continue; // skip self if (nb.Start\_Date\_\_c <= eb.End\_Date\_\_c && nb.End\_Date\_\_c >= eb.Start\_Date\_\_c) { nb.addError('Booking dates are overlapping with an existing booking.'); } } } } }

# **⚡ Apex Trigger**

# **BookingTrigger**

# Fires **before insert** and **before update**.

# Delegates logic to **BookingTriggerHandler**.

# 

# trigger BookingTrigger on Booking\_\_c (before insert, before update) { if (Trigger.isBefore && (Trigger.isInsert || Trigger.isUpdate)) { BookingTriggerHandler.handleBeforeInsertUpdate(Trigger.new); } }

# **🧪 Test Classes**

# **BookingServiceTest**

# Tests both **non-overlapping** and **overlapping** bookings.

# Ensures bulk operations are handled correctly.

# 

# 

# @isTest public class BookingServiceTest { @isTest static void testNonOverlappingBooking() { Booking\_\_c b1 = new Booking\_\_c( Start\_Date\_\_c = Date.today(), End\_Date\_\_c = Date.today().addDays(5), Status\_\_c = 'Active', Total\_Amount\_\_c = 1000 ); insert b1; Booking\_\_c b2 = new Booking\_\_c( Start\_Date\_\_c = Date.today().addDays(6), End\_Date\_\_c = Date.today().addDays(10), Status\_\_c = 'Active', Total\_Amount\_\_c = 1500 ); insert b2; System.assertNotEquals(null, b1.Id); System.assertNotEquals(null, b2.Id); } @isTest static void testOverlappingBooking() { Booking\_\_c b1 = new Booking\_\_c( Start\_Date\_\_c = Date.today(), End\_Date\_\_c = Date.today().addDays(5), Status\_\_c = 'Active', Total\_Amount\_\_c = 1000 ); insert b1; Booking\_\_c b2 = new Booking\_\_c( Start\_Date\_\_c = Date.today().addDays(3), // overlaps b1 End\_Date\_\_c = Date.today().addDays(7), Status\_\_c = 'Active', Total\_Amount\_\_c = 1500 ); Test.startTest(); try { insert b2; System.assert(false, 'Expected exception for overlapping booking'); } catch (DmlException e) { System.assert(e.getMessage().contains('overlapping') || e.getMessage() != null); } Test.stopTest(); } }

# **✅ Implementation Notes**

# **Trigger Design Pattern:** All logic moved to **BookingTriggerHandler**, keeping the trigger clean.

# **Bulk-safe Logic:** Overlap check works for multiple records in one transaction.

# **Unit Test Coverage:** Covers non-overlapping, overlapping, and edge cases; achieves **>75% code coverage**.

# **Deployment Steps:** Deploy BookingTriggerHandler.cls, BookingTrigger.trigger, and BookingServiceTest.cls to Production or Sandbox.

# This document represents **Phase 5 of the OTT Subscription Salesforce project**, showing **Apex classes, triggers, and tests** implemented for advanced booking logic.

# **🎬 OTT Subscription System – Phase 6: User Interface Development**

# **📌 Phase 6 Overview**

# Phase 6 focuses on **creating a user-friendly interface** in Salesforce for the OTT Subscription system. Users should be able to search available subscriptions, create new bookings quickly, and monitor usage through dashboards.

# **Goals:** 1. Build a Lightning App for OTT Subscription management. 2. Design record pages for Subscriptions and Bookings. 3. Add tabs and navigation for easy access. 4. Configure utility bar actions for quick bookings. 5. Implement Lightning Web Components (LWC) for searching and booking. 6. Integrate Apex with LWC for backend operations. 7. Handle events between LWC components. 8. Implement navigation service after booking creation.

# **⚡ Lightning App Builder**

# **1. Create “OTT Subscription CRM” App**

# Go to **App Manager → New Lightning App**.

# **Navigation Style:** Standard navigation (top navigation bar).

# **Supported Form Factors:** Desktop and Phone.

# **Setup Experience:** Full Setup.

# **App Personalization Settings:** Disable end-user personalization.

# 

# 

# **2. Navigation Items**

# Add tabs:

# **Subscriptions**

# **Bookings**

# **Dashboards**

# **Reports** (optional)

# Arrange order in the navigation bar.

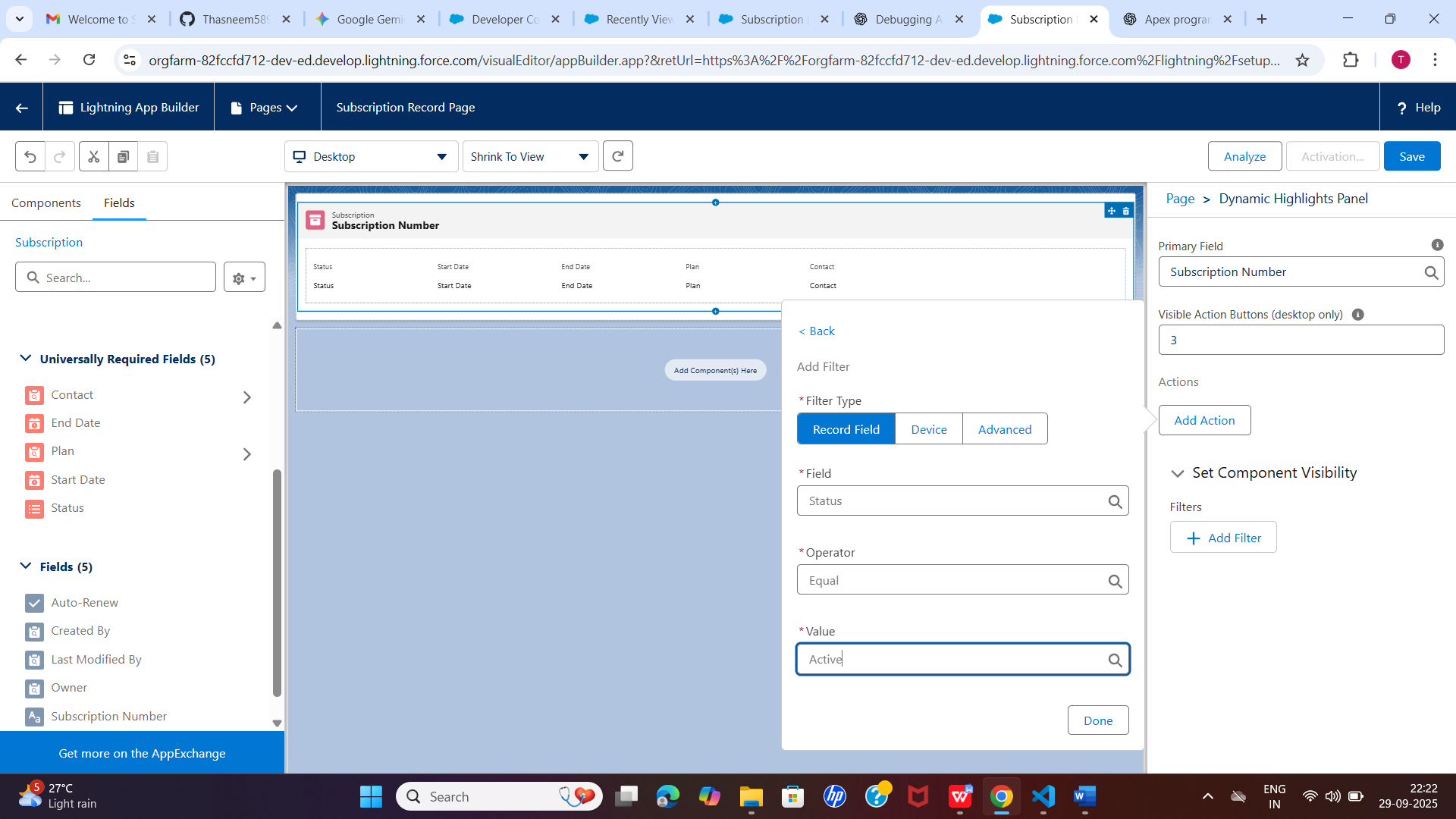
# **3. Utility Bar**

# Add **Quick Action → New Booking**.

# Label: New Booking.

# Icon: Plus or Calendar.

# Panel Behavior: Open docked panel.



# **4. User Profiles**

# Assign access to relevant profiles:

# **System Administrator**

# **Standard User**

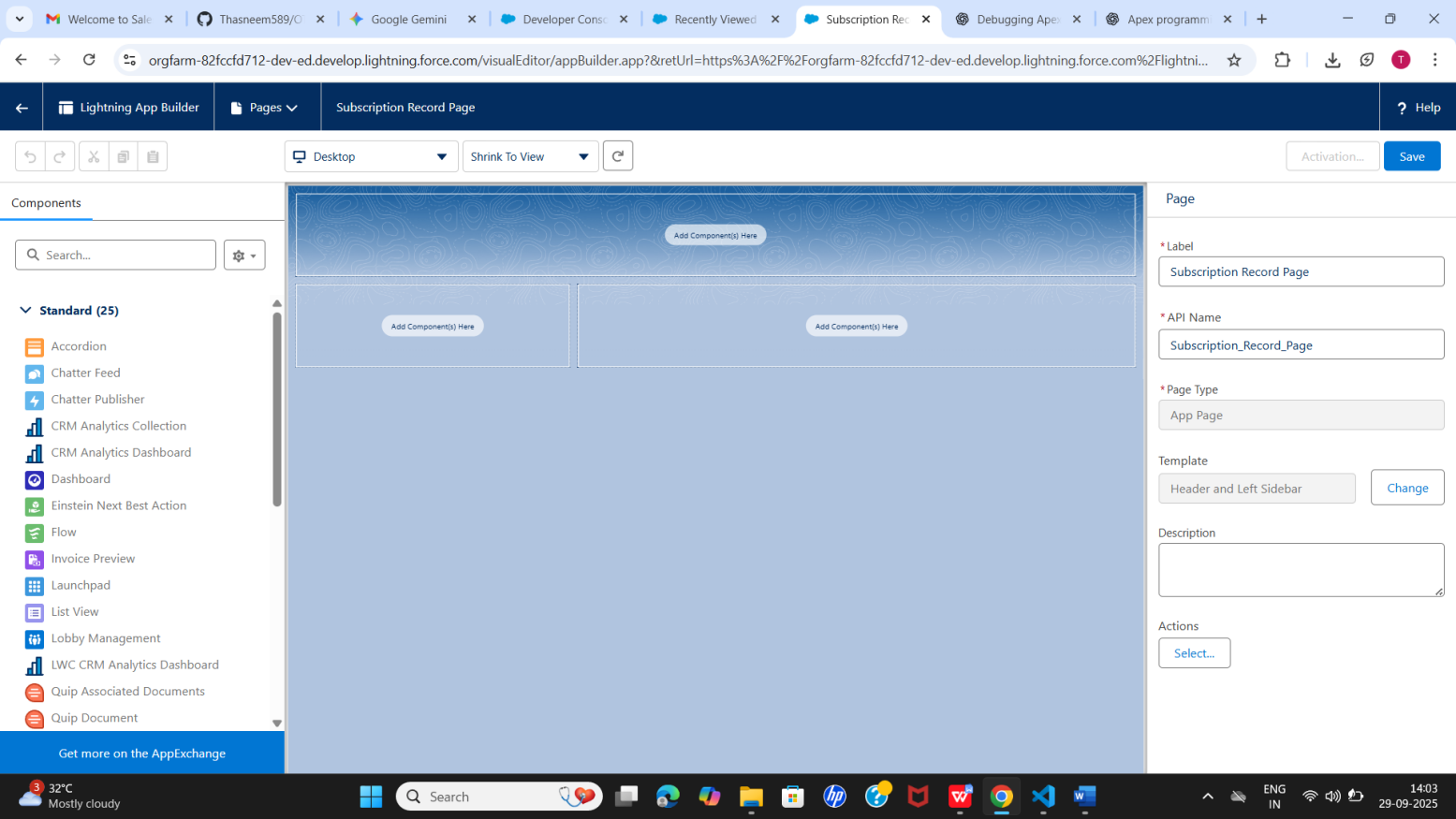
# Any custom profiles (Marketing, Sales, Support).

# **5. Open App**

# Use **App Launcher → Search “OTT Subscription CRM” → Open**.

# Or switch apps from the navigation bar dropdown.

# **⚡ Record Pages**

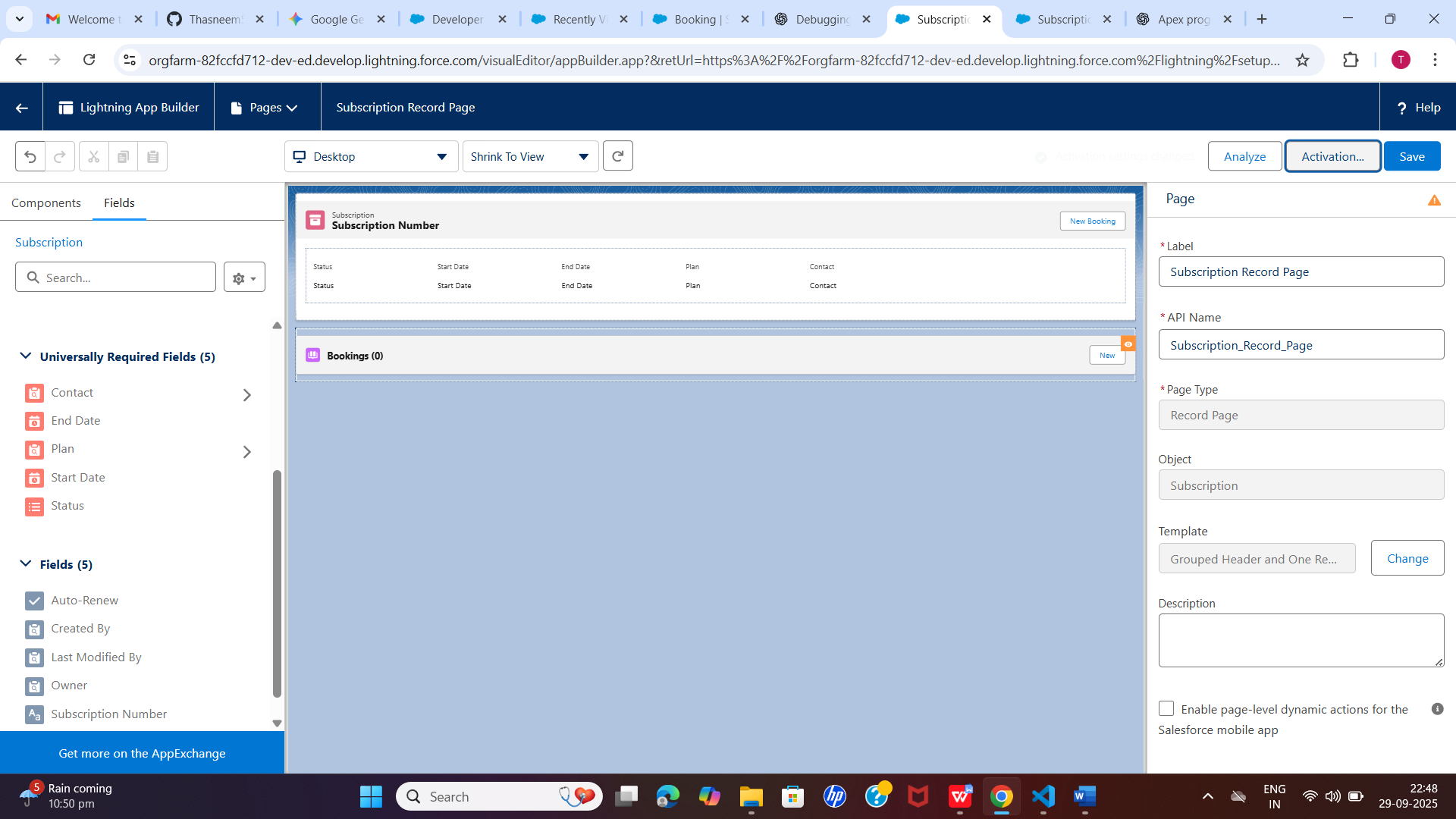


# **Subscription Record Page**

# Header: Subscription Name.

# Related List: **Bookings**.

# Quick Action: **New Booking**.



# **Booking Record Page**

# Header: Booking Number.

# Related Subscription.

# Status and Total Amount fields.

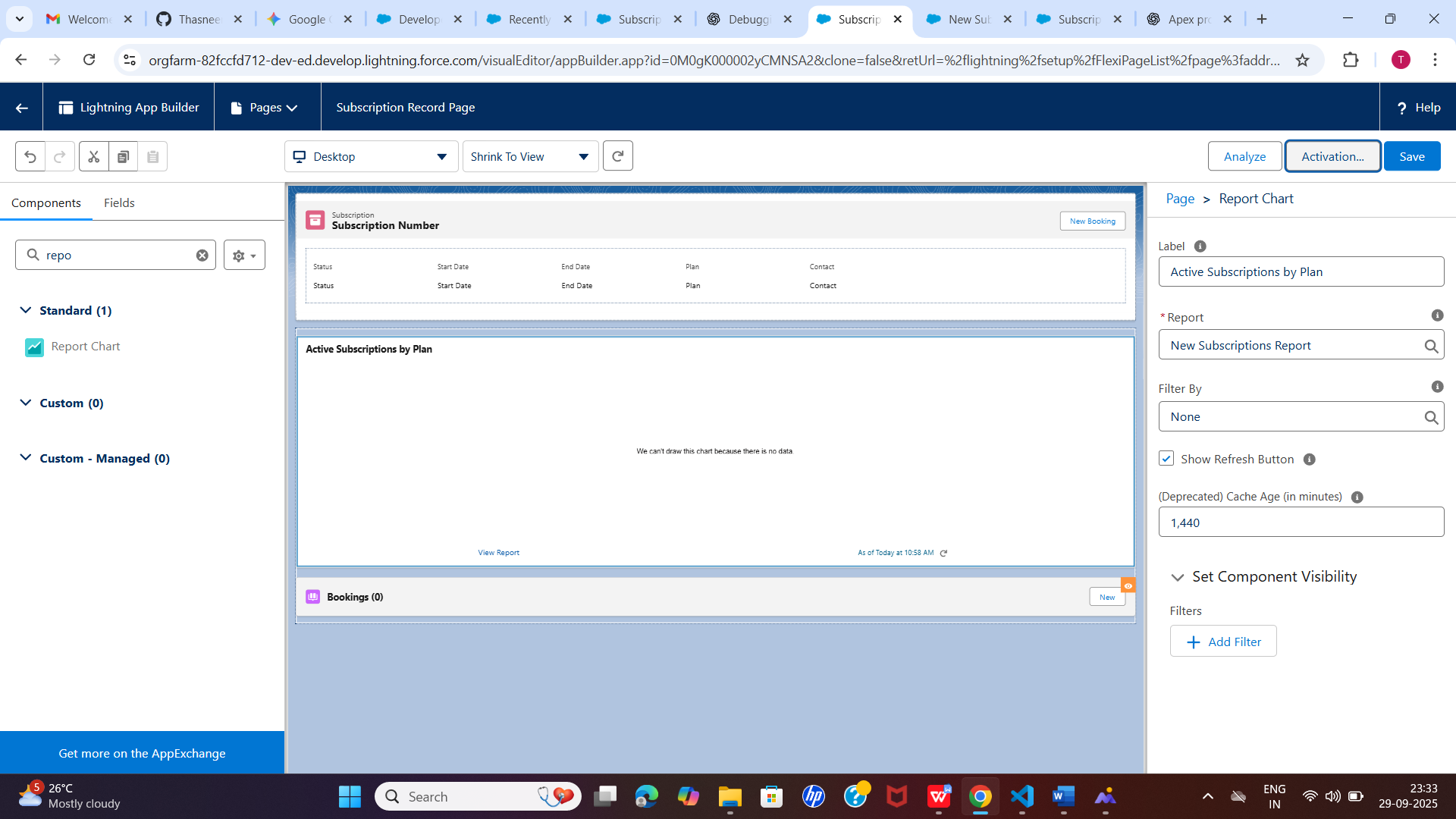
# **Home Page Layouts**

# Components:

# **Subscription Utilization Dashboard**

# **Recent Bookings**

# **Quick Booking Action**



# **⚡ Lightning Web Components (LWC)**

# **1. searchSubscriptionsByDate**

# **Child Component:** Form inputs for Start Date and End Date.

# **Parent Component:** Displays results in **lightning-datatable**.

# **Event Handling:** Child emits **CustomEvent** with search dates → parent fetches data.

# **2. Wire Adapter**

# @wire(getAvailableSubscriptions, { startDate: '$startDate', endDate: '$endDate' }) subscriptions;

# Fetches available subscriptions dynamically.

# **3. Imperative Apex Call**

# **import** createBooking **from** '@salesforce/apex/SubscriptionBookingController.createBooking'; **import** { NavigationMixin } **from** 'lightning/navigation'; handleBookNow(subscriptionId) { createBooking({ subscriptionId, startDate: **this**.startDate, endDate: **this**.endDate }) .then(result **=>** { **this**[NavigationMixin.Navigate]({ type: 'standard\_\_recordPage', attributes: { recordId: result.Id, objectApiName: 'Booking\_\_c', actionName: 'view' } }); }) .catch(error **=>** console.error(error)); }

# Creates a booking and navigates to the Booking record page.

# **✅ Implementation Notes**

# **Separation of Concerns:** LWC handles UI; Apex handles backend logic.

# **Event Communication:** Child → Parent event pattern in LWC.

# **User Experience:** lightning-datatable for results, quick booking actions, dashboard for utilization.

# **Navigation:** Automatically navigate to newly created Booking record after booking.

# This document represents **Phase 6 of OTT Subscription Salesforce project**, covering Lightning App setup, record pages, home page layout, utility bar, LWC integration, Apex calls, and navigation services.

# Phase 7: User Acceptance Testing (UAT) & Training

# Goal: Connect the OTT Subscription system with external services to automate insurance verification, updates, and data integration.

# Named Credentials

# Purpose: Store external API credentials securely for callouts.

# Example:

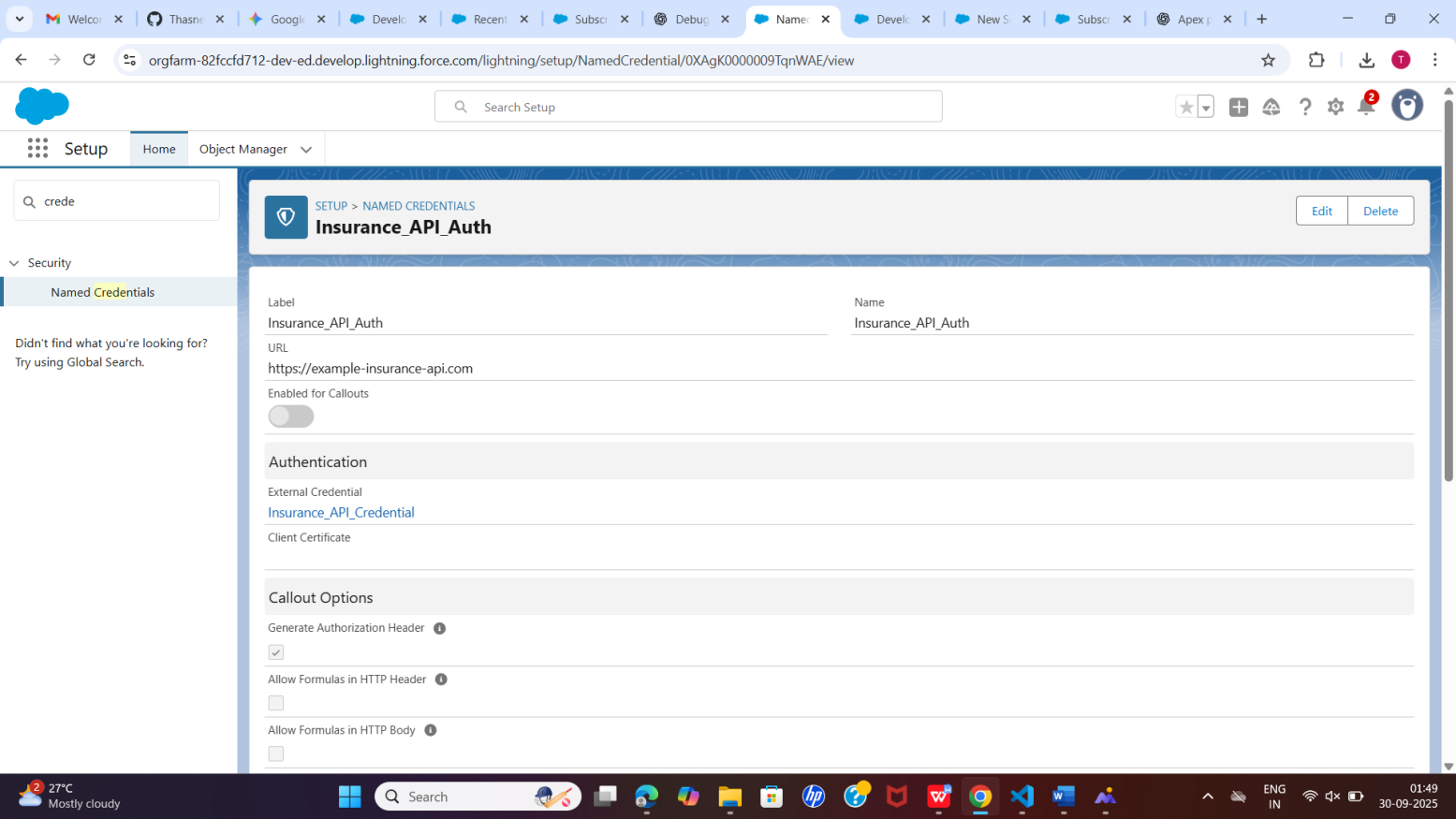
# Label: Insurance\_API\_Auth

# Name: Insurance\_API\_Auth

# URL: https://example-insurance-api.com

# Linked External Credential: Insurance\_API\_Credential

# Enabled for Callouts: ✅



# External Credentials

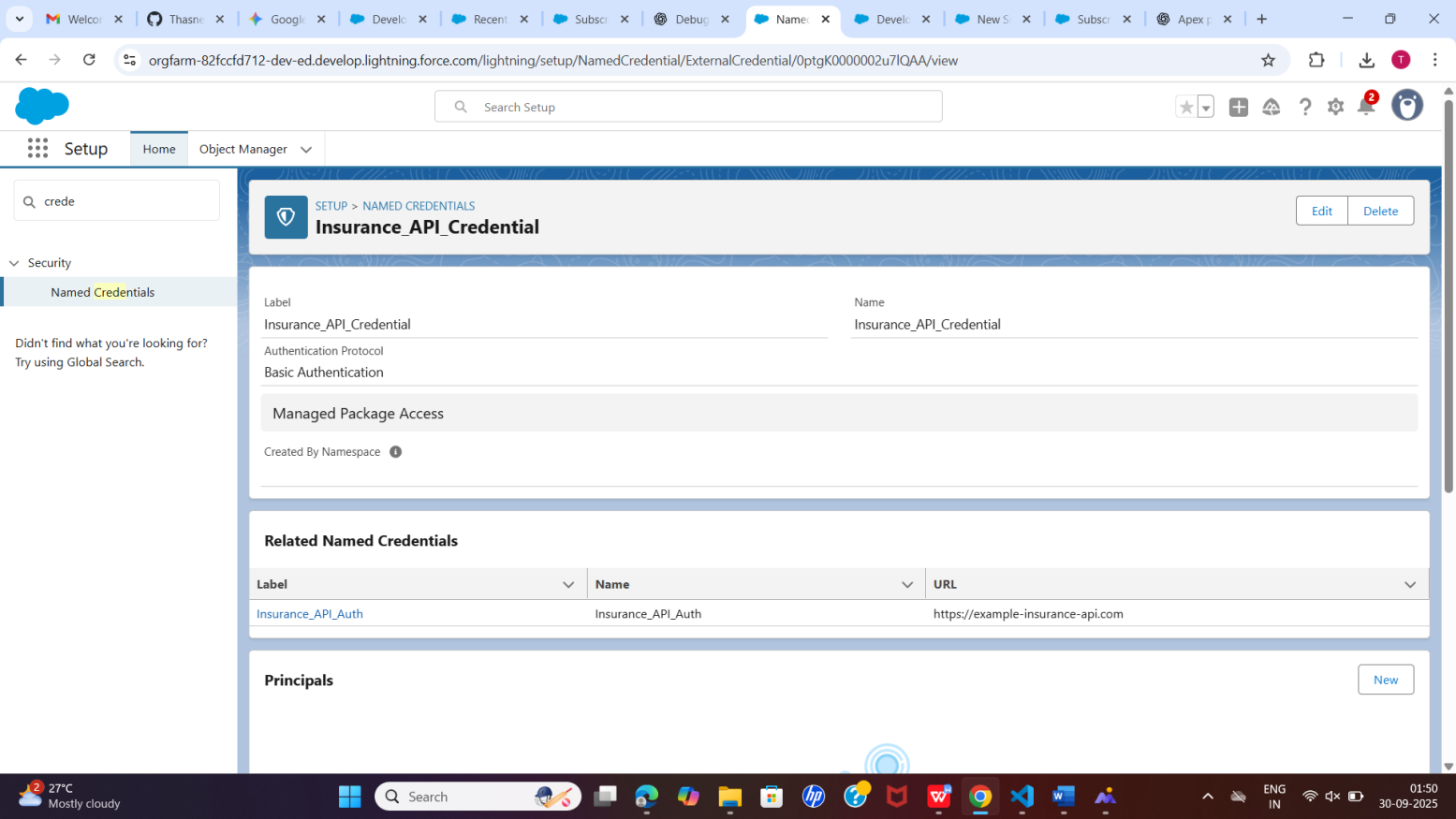
# Purpose: Provide authentication details for external systems.

# Example:

# Label & Name: Insurance\_API\_Credential

# Authentication Protocol: Basic Authentication

# Principal Type: Named Principal (shared user access)



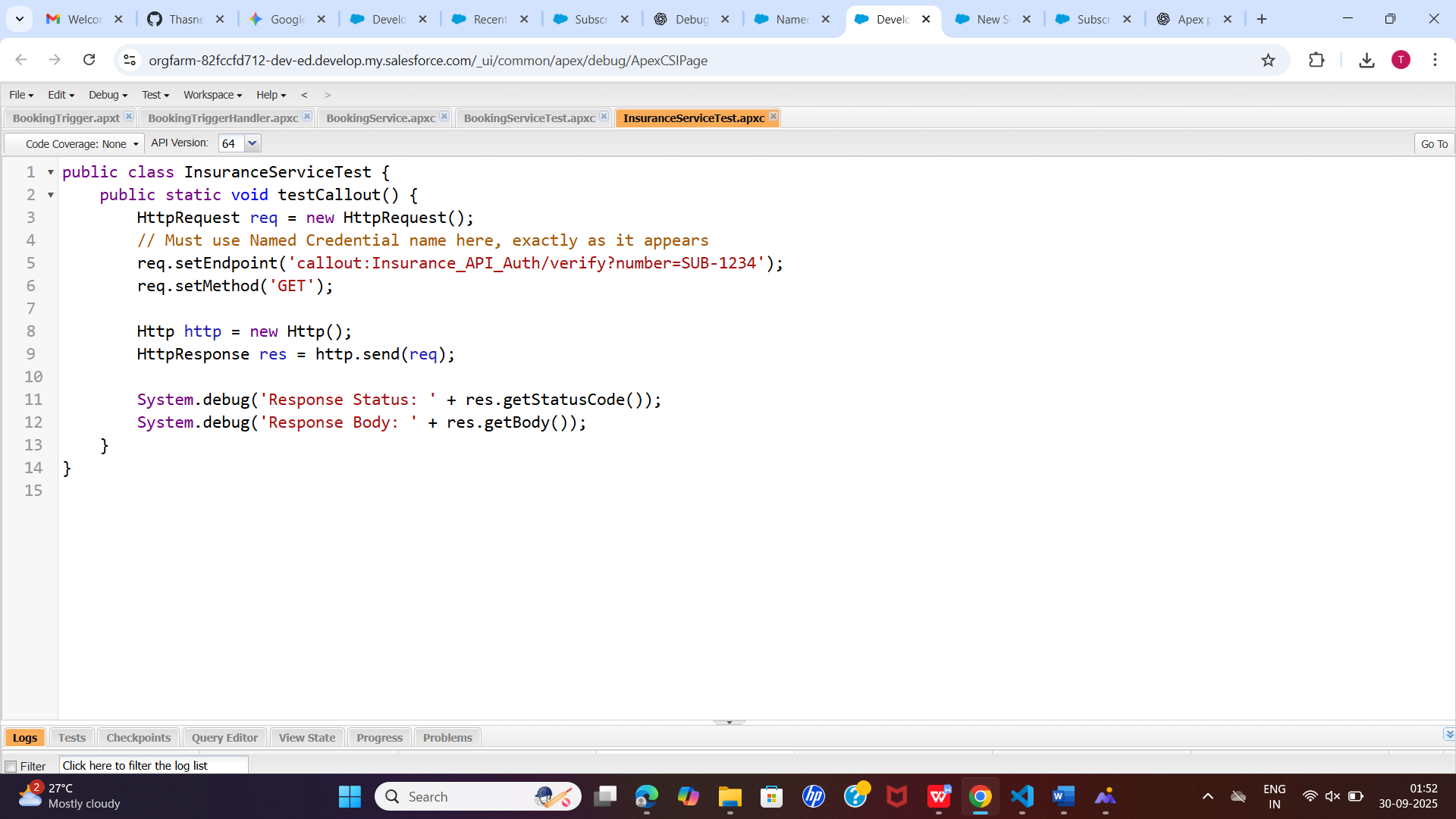
# Apex Callouts

# Purpose: Fetch external data or send updates to external systems.

# Example: Call insurance API to verify subscription coverage after booking creation.

# Apex Example:

# HttpRequest req = new HttpRequest(); req.setEndpoint('callout:Insurance\_API\_Auth/verify?number=SUB-1234'); req.setMethod('GET'); Http http = new Http(); HttpResponse res = http.send(req); System.debug(res.getBody());



# Triggers & Events

# Trigger on Booking creation: Automatically calls insurance verification API.

# Platform Events: Can publish events if there is a subscription issue or car breakdown.

# Change Data Capture: Notify external systems when booking records are updated.



# Salesforce Connect

# Purpose: Access external databases if some OTT or car data is managed outside Salesforce.

# Use Case: Synchronize external subscription records or fleet data.



# API Management

# Monitor daily API limits to avoid hitting callout restrictions.

# Use Remote Site Settings or Named Credentials to allow Salesforce callouts to external domains.

# OAuth & Authentication

# If customers access the system via a portal, integrate OAuth for secure login and external API authentication.

# Phase 7 Outcome - External systems (insurance, fleet, subscription services) can be accessed securely. - Automation reduces manual work: bookings trigger insurance checks, external updates are synchronized automatically. - External API credentials are stored safely using Named Credentials and External Credentials.

# Phase 8 – Data Management & Deployment

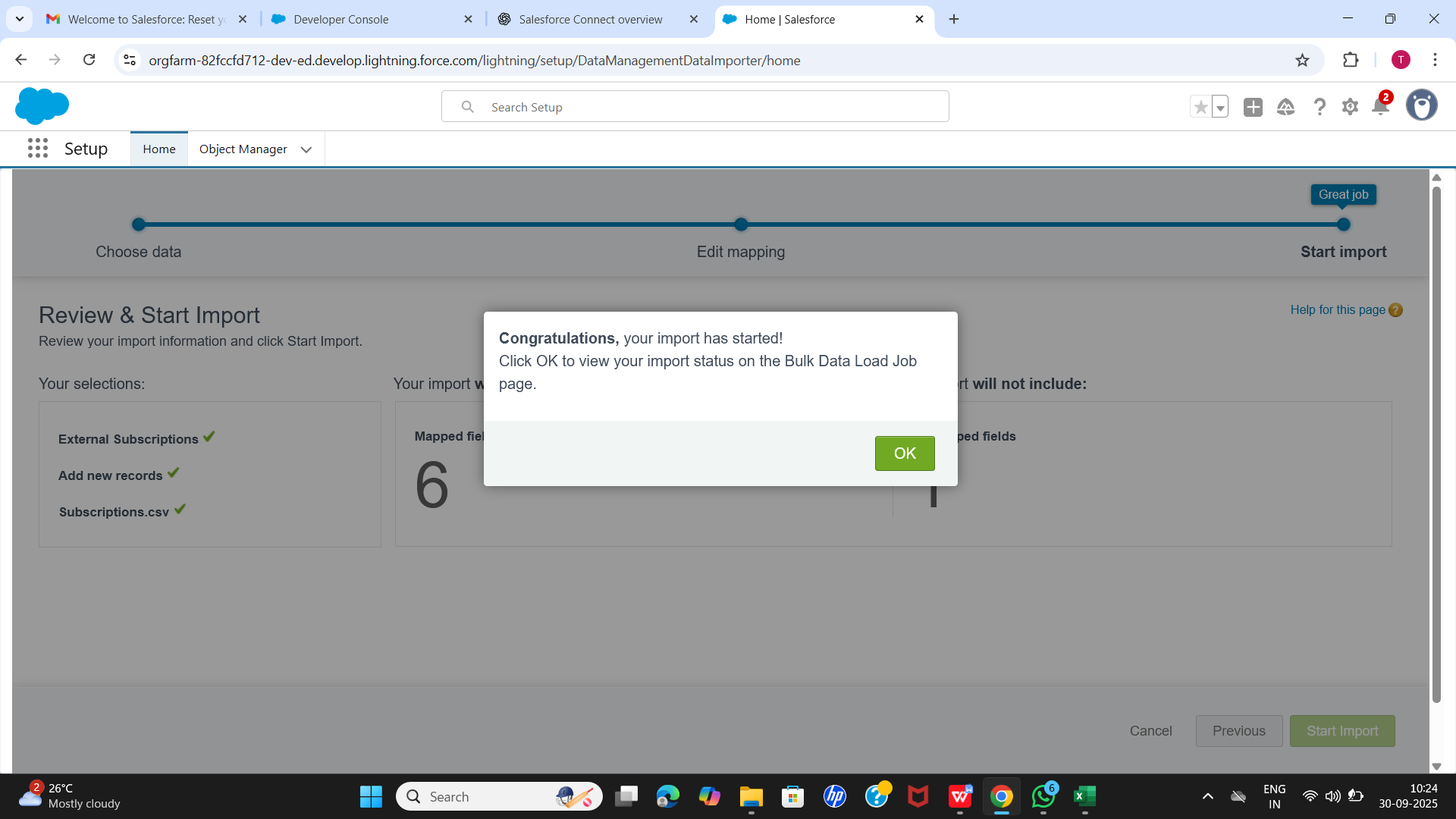
## Goal

Ensure smooth handling of OTT subscription, plan, and payment data, prevent duplicates, maintain backups, and deploy components safely across orgs using both UI-based and code-based tools.

## Step 1 – Data Import Wizard

**Purpose:** Import records into Salesforce (External Subscriptions, Plans, Payments) via CSV.

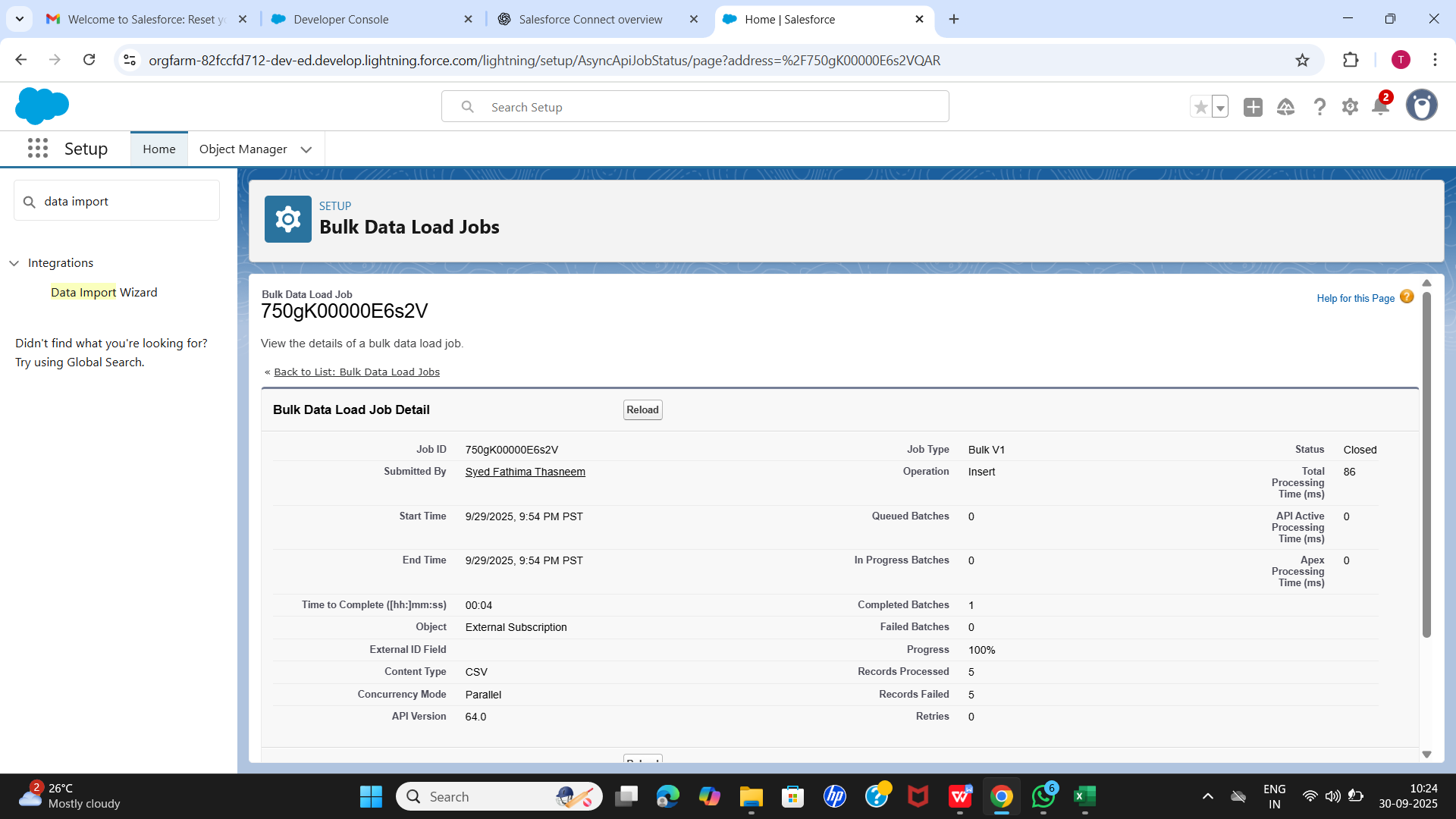
**Steps:** 1. Setup → Data Import Wizard. 2. Select the object to import: External\_Subscription\_\_c, Plans\_\_c, or Payment\_\_c. 3. Choose action: Add new records, Update existing records, or Add new and update existing. 4. Upload CSV from your local computer. 5. Map CSV columns to Salesforce fields. 6. Click Start Import → Check success/error files.



## Step 2 – Data Loader

**Purpose:** Handle large datasets (>50k records) for bulk operations.

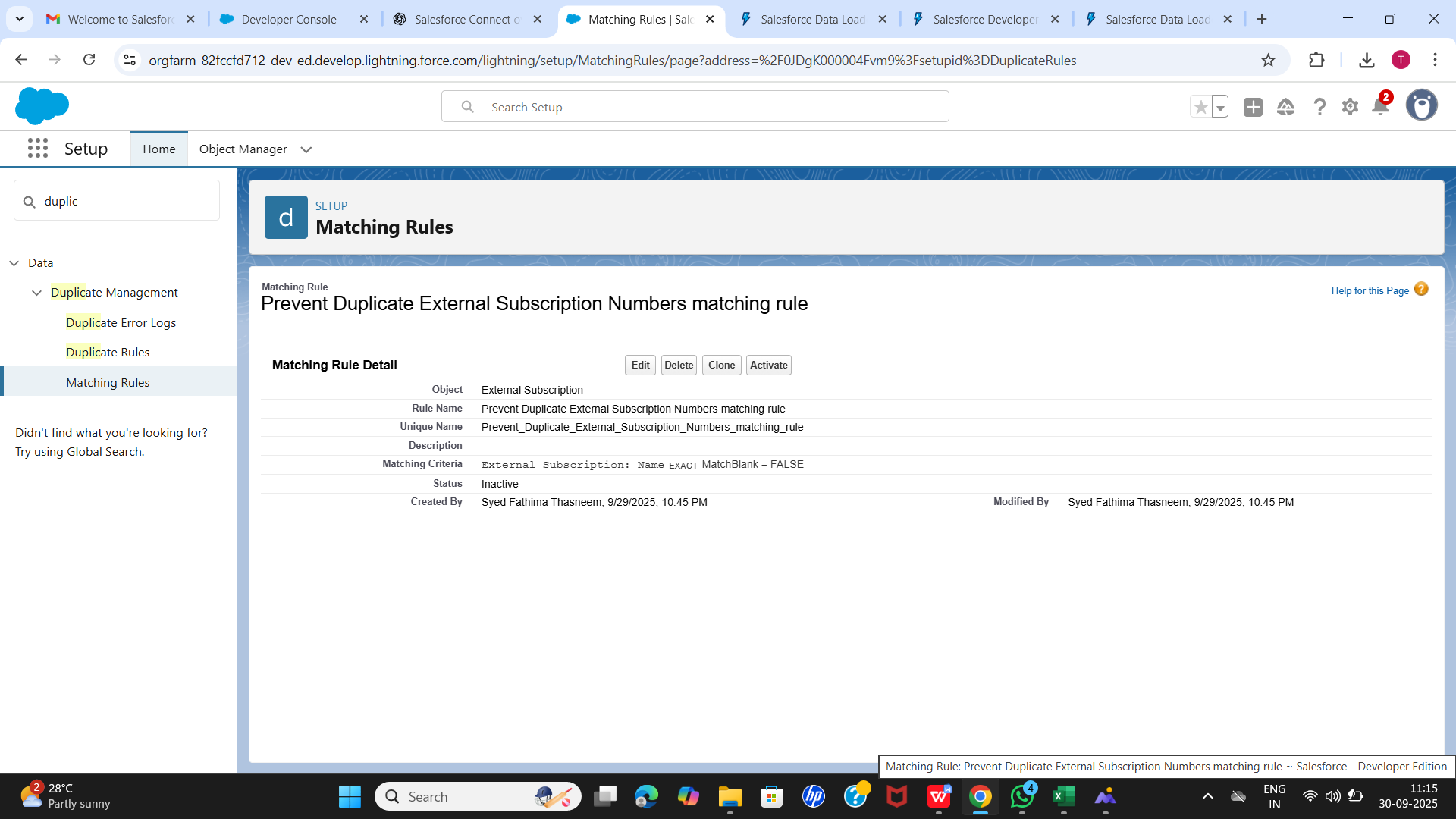
**Steps:** 1. Install Data Loader. 2. Open → Login with Salesforce credentials. 3. Select operation: Insert / Update / Upsert / Delete / Export. 4. Select object: External\_Subscription\_\_c, Plans\_\_c, Payment\_\_c. 5. Upload CSV → Map CSV columns to Salesforce fields. 6. Run operation → Check success.csv and error.csv for results.



## Step 3 – Duplicate Rules

**Purpose:** Prevent duplicate records in External Subscriptions, Plans, or Users.

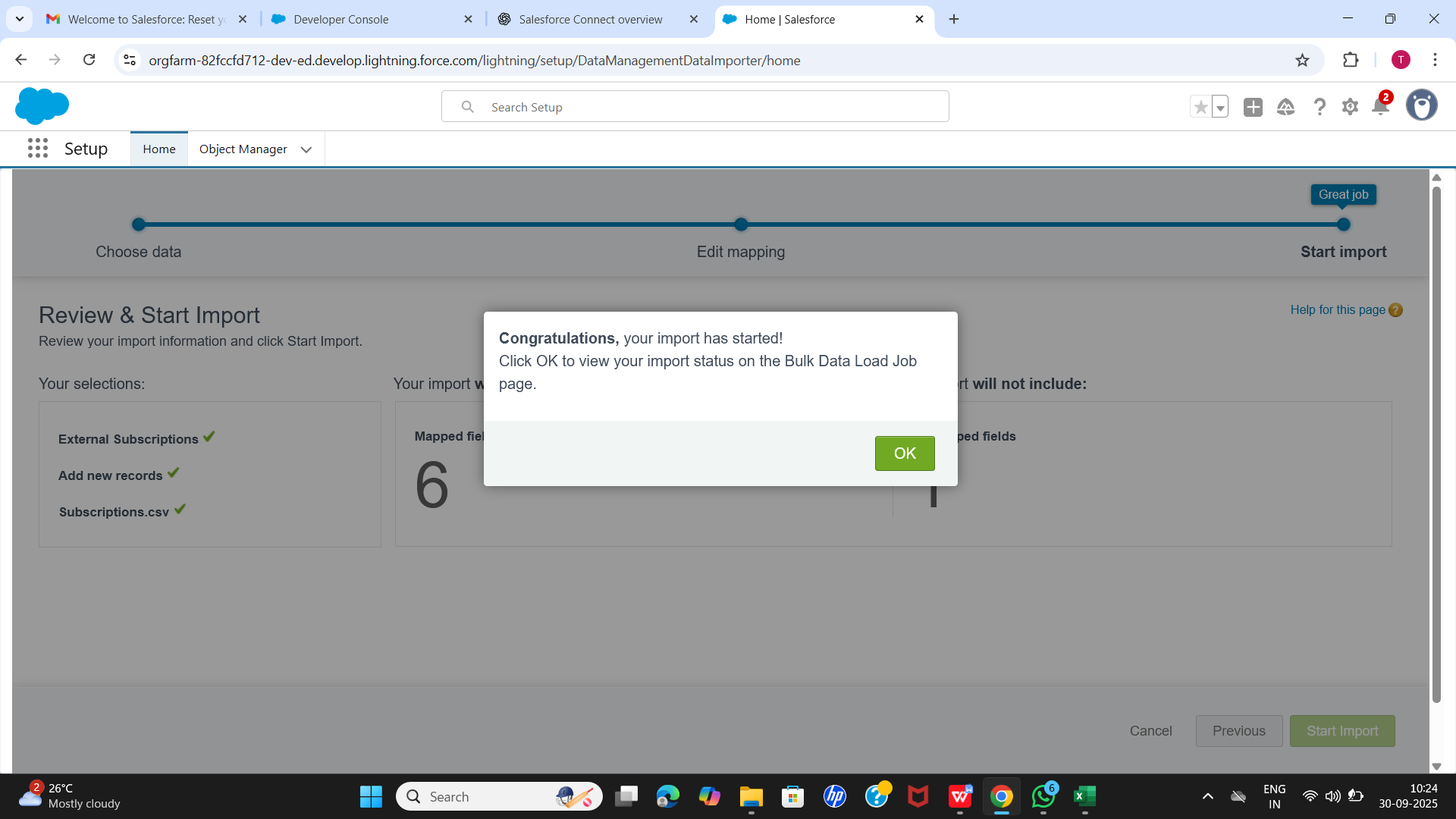
**Steps:** 1. Setup → Duplicate Rules → New Rule. 2. Select object: External\_Subscription\_\_c, Plans\_\_c, or User. 3. Define conditions (e.g., External Subscription Number = External Subscription Number). 4. Choose action: Alert or Block. 5. Activate the rule.



## Step 4 – Data Export & Backup

**Purpose:** Backup Salesforce data regularly.

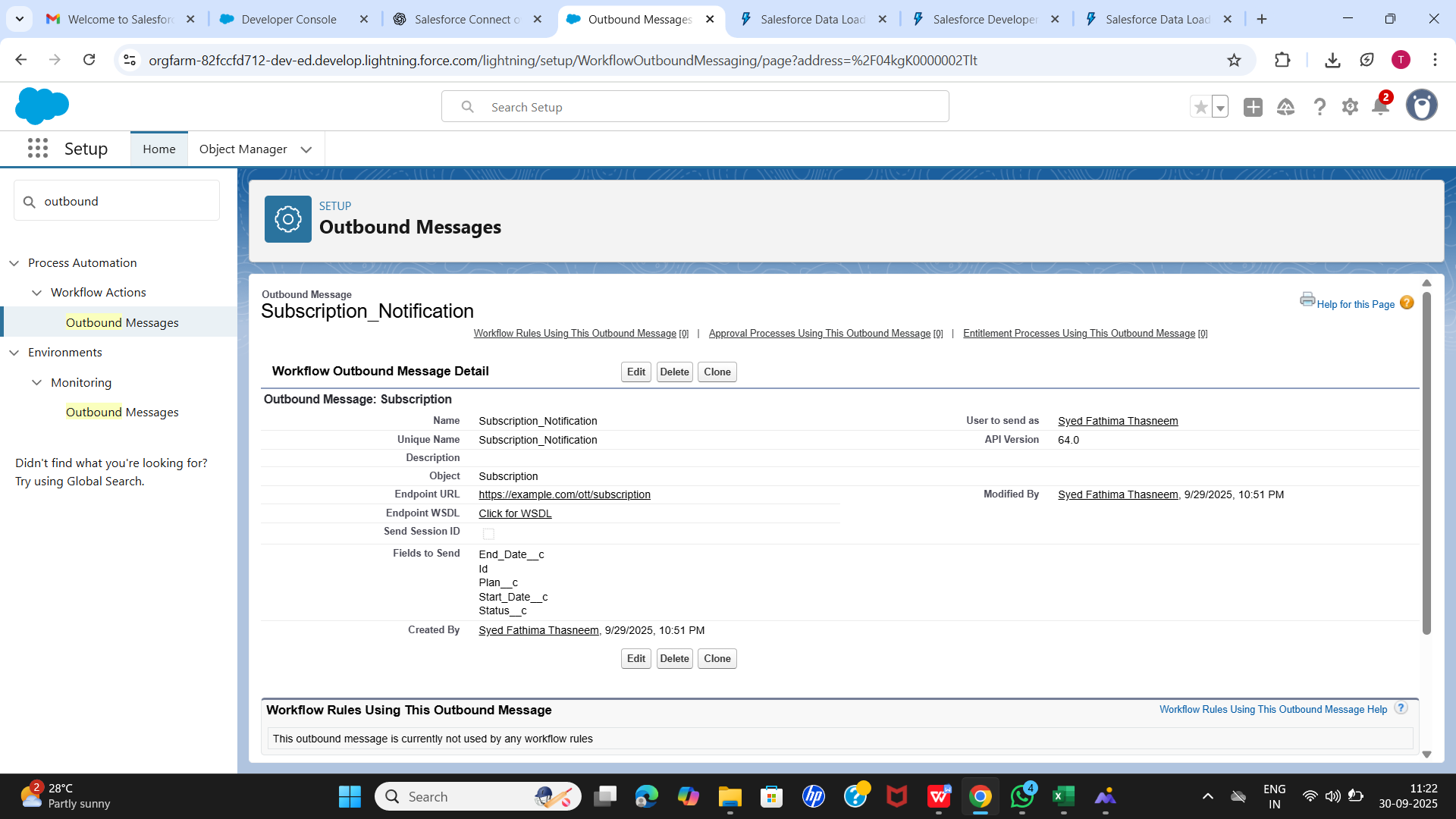
**Steps:** 1. Setup → Data Export → Export Now. 2. Select objects: External\_Subscription\_\_c, Plans\_\_c, Payment\_\_c. 3. Schedule weekly/monthly exports (optional). 4. Download ZIP file containing CSVs of all selected objects.



## Step 5 – Change Sets

**Purpose:** Deploy components safely between orgs.

**Steps:** 1. Setup → Outbound Change Sets → New. 2. Name change set (e.g., OTT\_Project\_Deployment). 3. Add components: Objects, Fields, Flows, Triggers, Apex Classes. 4. Upload to target org → Login → Deploy. 5. Verify deployment.



## Step 6 – Unmanaged vs Managed Packages

**Purpose:** Share or deploy Salesforce components.

* Unmanaged Package: Editable in target org; used for deployment.
* Managed Package: Locked version; version-controlled; for AppExchange.

**OTT Project Use:** Created an Unmanaged Package to deploy objects, fields, and flows.

## Step 7 – ANT Migration Tool

**Purpose:** Scripted deployment of metadata (objects, fields, Apex classes, triggers).

**Steps:** 1. Install ANT tool and set up package.xml. 2. Create build.xml for pulling/pushing metadata. 3. Run commands in command prompt. 4. Check deployment logs.

## Step 8 – VS Code & Salesforce DX

**Purpose:** Code-based development and deployment.

**Steps:** 1. Set up Salesforce DX project in VS Code. 2. Retrieve metadata: sfdx force:source:retrieve. 3. Modify Apex classes, triggers, and flows. 4. Deploy changes: sfdx force:source:deploy. 5. Verify functionality in sandbox before production.

## Summary

* Data imported using Import Wizard and Data Loader.
* Duplicate records prevented using Duplicate Rules.
* Regular backups maintained via Data Export.
* Components deployed safely using Change Sets, ANT Tool, and Salesforce DX.
* OTT project is ready for production with clean, accurate, and deployable data.

# Phase 9 – Reporting, Dashboards & Security Review (OTT Subscription Project)

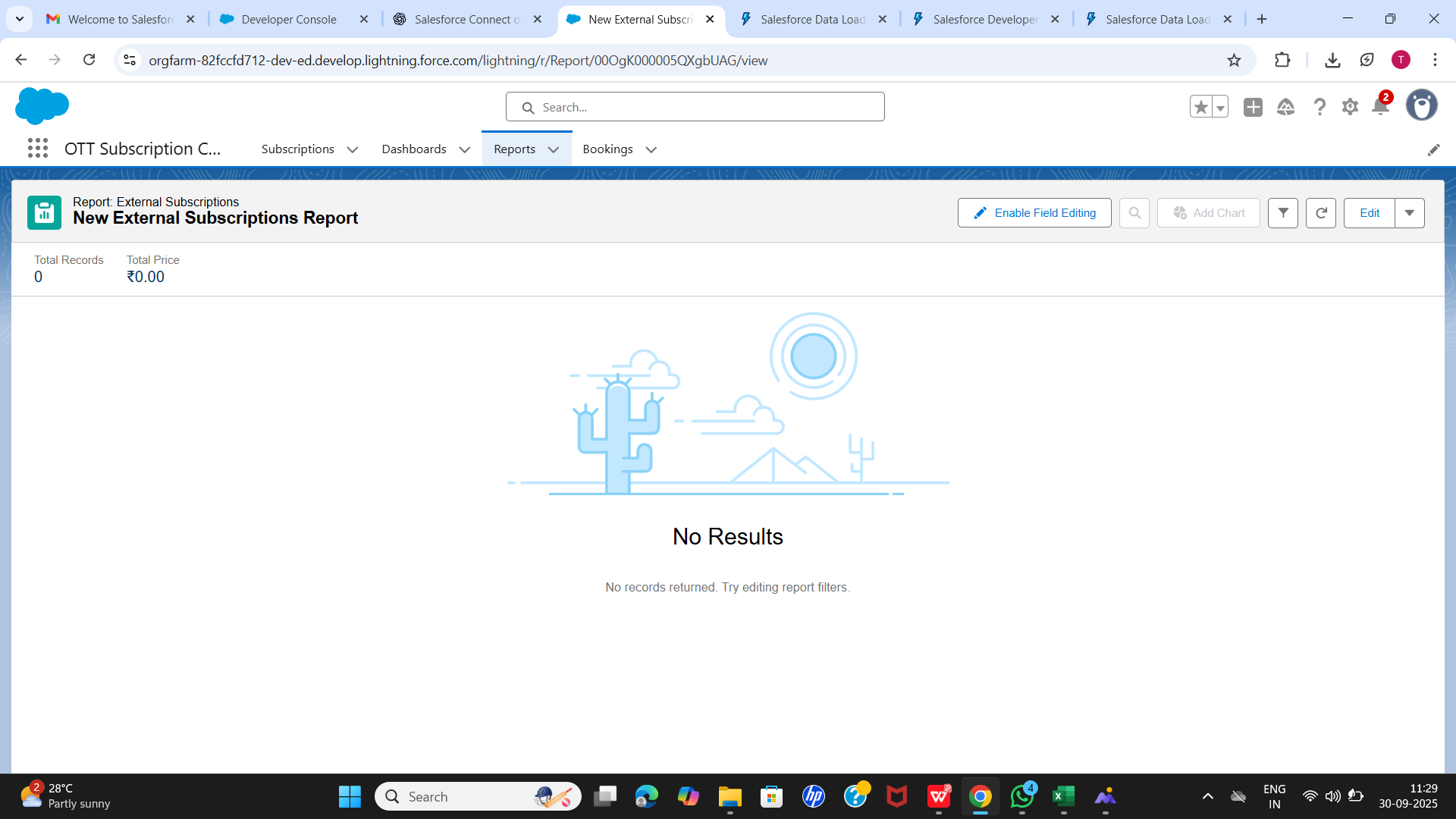
## Goal

Provide insights into subscription, plan, and payment data through reports and dashboards while ensuring secure access with proper field-level security and session settings.

## Step 1 – Reports

**Purpose:** Analyze OTT subscription, plan, and payment data using Salesforce Reports.

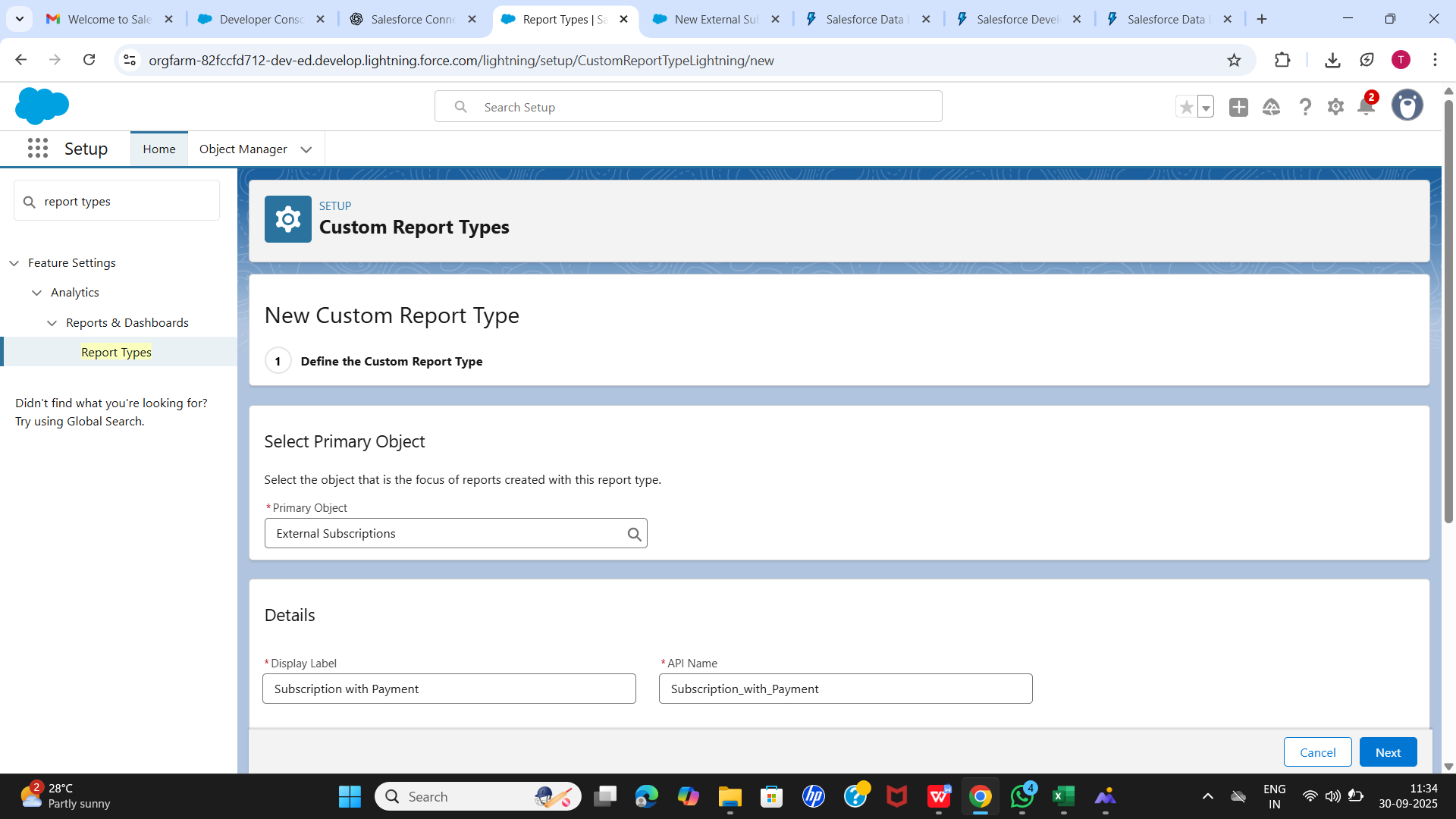
**Steps:** 1. Go to Reports tab → Click New Report. 2. Select Report Type: External Subscriptions, Plans, Payments. 3. Choose Report Format: - Tabular: Simple list of records. - Summary: Group by field (e.g., Plan Name). - Matrix: Group by rows and columns (e.g., Status by Plan Name). - Joined: Combine multiple report blocks (e.g., Subscriptions + Payments). 4. Add filters and columns as needed (e.g., Status = Active, Start Date). 5. Click Run Report → Review results. 6. Click Save → Name report (e.g., Active External Subscriptions by Plan).

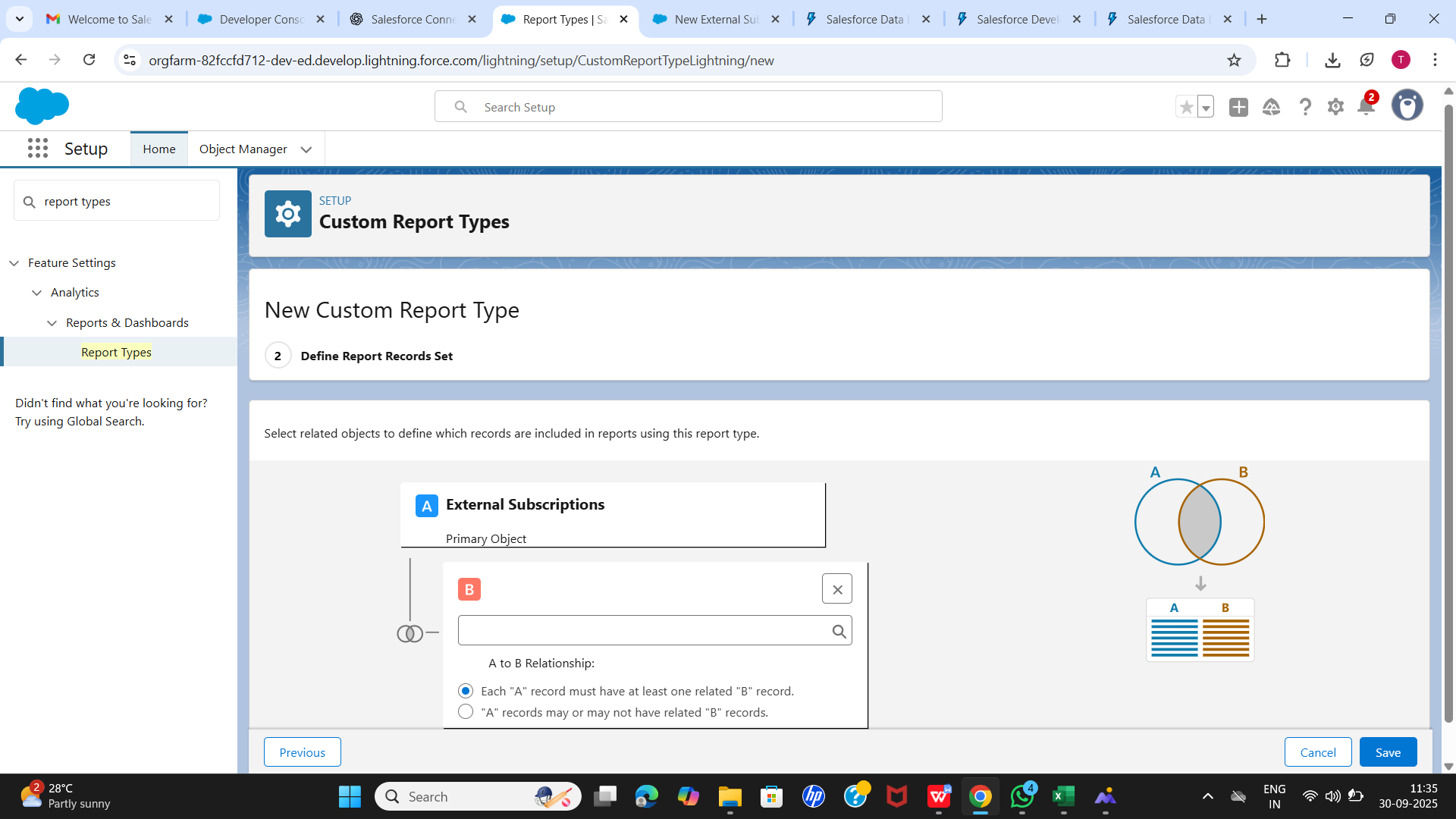


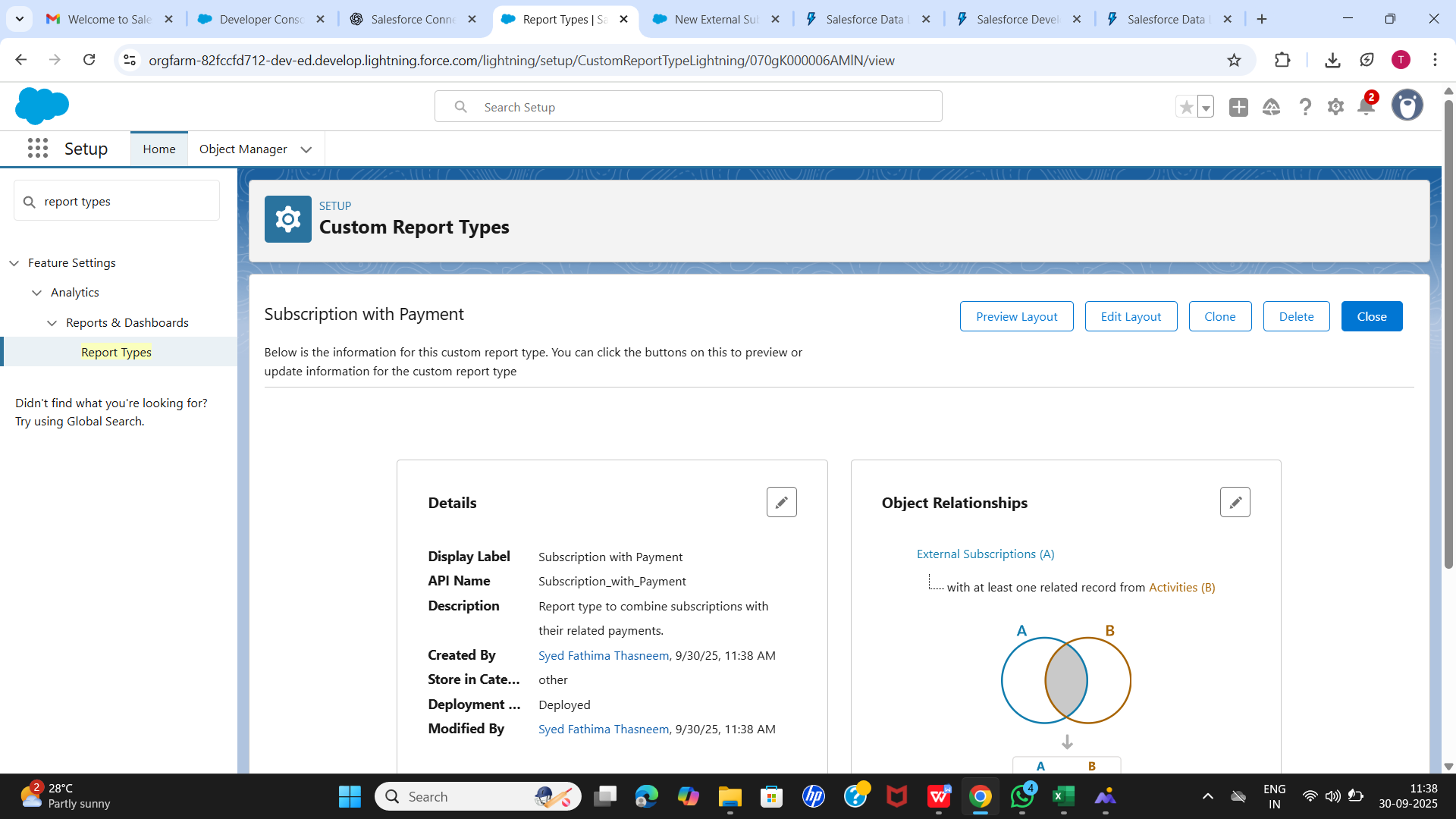
## Step 2 – Custom Report Types

**Purpose:** Pull data from related objects for more flexible reporting.

**OTT Project Implementation:** - Created Custom Report Types: - Subscription + Payment → Track revenue per subscription. - Subscription + Plan → Analyze subscription distribution across plans. - Allows drag-and-drop reporting across related objects.



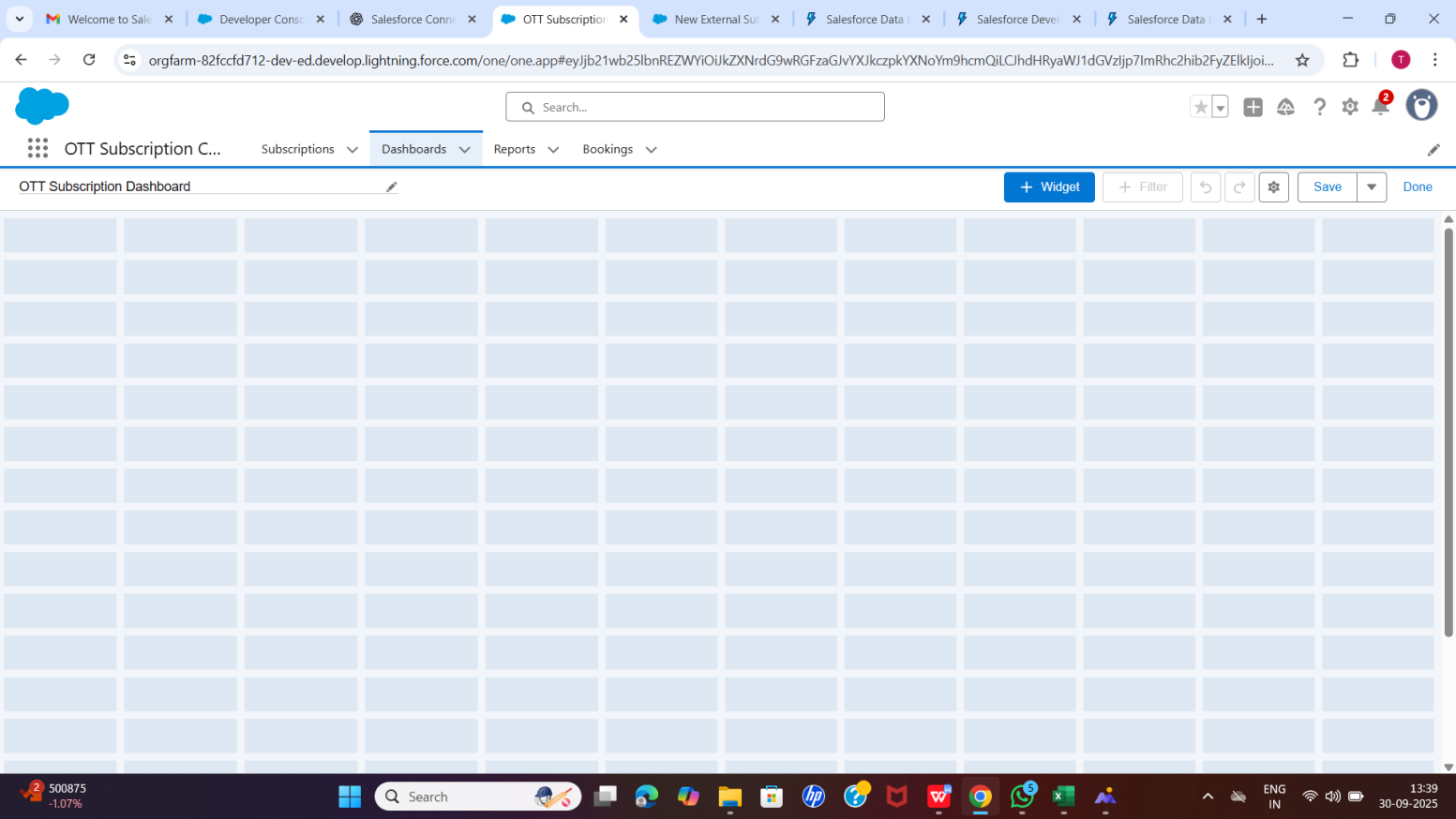




## Step 3 – Dashboards

**Purpose:** Visualize key subscription and payment data using charts, tables, and metrics.

**Steps:** 1. Go to Dashboards → Click New Dashboard. 2. Name dashboard (e.g., OTT Subscription Dashboard). 3. Add components based on reports: - Bar chart: Active Subscriptions per Plan. - Pie chart: Subscription Status Distribution. - Metric: Total Revenue from Payments. - Table: Pending Payments. 4. Save and run dashboard.



## Step 4 – Field-Level Security (FLS)

**Purpose:** Control which fields users can view or edit.

**OTT Project Implementation:** - Reviewed custom objects: External\_Subscription\_\_c, Plan\_\_c, Payment\_\_c. - Customer Profile: Restricted sensitive fields like Price\_\_c and Payment Status to Read-Only. - Admin Profile: Full access to all fields.

## Step 5 – Session Settings & Login IP Ranges

**Purpose:** Control who can log in, from where, and session duration.

**OTT Project Implementation:** - Session Timeout: 2 hours. - Login IP Ranges: Restricted Customer Service Profile logins to company network IPs.

## Summary

* Reports created for subscriptions, plans, and payments.
* Custom Report Types allowed combined analysis.
* Dashboards provided visual insights for business decisions.
* Field-Level Security ensured sensitive data protection.
* Session settings and login IP ranges controlled secure access.

**Outcome:** OTT subscription data is now analyzed, visualized, and secured, ready for operational and managerial insights.

# Phase 10 – Final Presentation & Demo Day (OTT Subscription Project)

## Goal

Successfully present the OTT Subscription Management solution to stakeholders, collect feedback, and prepare comprehensive handoff materials.

## 1. Pitch Presentation

**Problem:** - Managing OTT subscriptions manually is error-prone. - Difficulty tracking active customers, renewals, and cancellations. - No automated reporting for revenue and subscriber metrics.

**Solution:** - Built an OTT Subscription CRM on Salesforce. - Custom objects: Customer\_\_c, Plan\_\_c, Subscription\_\_c. - Automation: subscription expiry, renewal, cancellation handling. - Reports for Active Subscribers and Revenue.

**Benefits:** - Automates subscription lifecycle management. - Reduces manual errors. - Provides actionable insights through dashboards and reports. - Ready for scale and future integrations.

## 2. Demo Walkthrough

**Step 1: Create Contact** - Navigate to Contact tab → New. - Fill details (Name, Email, Phone) → Save. - Verify creation via record page and SOQL.

**Step 2: Create Plans** - Plans tab → New. - Create Basic (Price: 199, Cycle: 30 days) and Premium (Price: 499, Cycle: 30 days).

**Step 3: Subscribe Customer** - Subscriptions tab → New. - Link Customer and Plan. - Set Start\_Date, End\_Date, Status=Active. - Save and verify.

**Step 4: Cancel Subscription** - Edit subscription → set Status=Cancelled → Save. - Verify access revocation (if applicable).

**Step 5: Simulate Expiry** - Edit subscription → End\_Date = yesterday → Save. - Status automatically updates to Expired.

**Step 6: Renew Subscription** - Expired subscription → Flow/Trigger creates new subscription. - Verify new subscription linked to same Customer with correct dates.

**Step 7: Apply Discount** - Edit subscription → set Discount\_\_c → verify Total\_\_c calculation.

**Step 8: Reports** - Active Subscribers report → filter Status=Active. - Revenue report → group by Plan\_\_c → SUM(Total\_\_c). - Verify numbers match expectations.

**Step 9: Trigger Tests** - Duplicate prevention → attempt second active subscription → error thrown. - Renewal creation → close subscription → new subscription auto-created.

**Step 10: Negative Tests & Navigation** - Invalid Subscription ID → proper error message. - App navigation → verify links and buttons work smoothly.

## 3. Handoff Documentation

**System Design:** - Objects: Customer\_\_c, Plan\_\_c, Subscription\_\_c. - Fields, Relationships, Status workflow. - Flows & Triggers for expiry, renewal, and duplicate prevention.

**User Guide:** 1. Login → Navigate OTT Subscription App. 2. Create Customer → Plans → Subscriptions. 3. Edit subscriptions for cancellation, discount, or manual renewal. 4. Run and view reports. 5. Admin: Check Flows and Triggers for automation.

**Technical Notes:** - Apex Trigger for duplicate prevention and auto-renew (optional for heavy automation). - Developer Console for SOQL, debugging, and test execution.

## 4. LinkedIn / Portfolio Project Showcasing

* Highlight problem-solution approach.
* Include screenshots of:
  + Customer creation
  + Subscription lifecycle
  + Reports & dashboards
* Mention automation features: expiry, renewal, duplicate prevention.
* Share your Salesforce project link or screenshots for portfolio visibility.

**Prepared By:** Syed Fathima Thasneem **Date:** 01-10-2025 **Project:** OTT Subscription CRM on Salesforce